On-demand provisioning of long-tail services in distributed clouds

Piet Smet, Bart Dhoedt and Pieter Simoens
Department of Information Technology (INTEC)
Ghent University - iMinds
Ghent, Belgium

***Abstract*—We see a trend to design services as a suite of small service components instead of the typical monolithic nature of classic web services, which led to an increasing amount of long-tail services on the Internet. Deploying instances everywhere to achieve a fast response time results in high costs, especially when these services are used infrequently and remain idle most of the time. One way to avoid needless over-provisioning is to deploy instances on-demand but this requires every component to be available upon request arrival.**

We propose a placement algorithm to maximize the amount of clients we can serve on-demand using the Docker layered filesystem. Docker facilitates automated deployment of services in lightweight software containers, allowing almost instantaneous deployment. Our algorithm finds the optimal storage location for layers so we can retrieve all service layers, deploy a service instance and provide a first response to a request within the desired time. We solve this problem using integer linear programming (*ILP*) and present techniques to improve the scalability of ILP while minimizing the performance loss. Results show that our approximation performs better with large scale problems than the classic ILP case.

***Keywords*-on-demand, provisioning, Docker, long-tail, services**

I. Introduction

In the last few years we see an increasing trend to build services as a suite of small components with a very specific functionality which can be deployed separately. This microservice architecture [1] led to an increase in small services on the Internet which are used infrequently. To meet the latency requirements, services are often deployed on distributed clouds in the edge network, also called Fog Computing [2]. Pre-deploying service instances across the edge network is economically infeasible and limits the number of services we can offer due to limited resources, yet we still wish to guarantee the desired response time.

On-demand provisioning is an interesting method to avoid cluttering edge devices with small service instances which are rarely used. Rather than pre-deploying idle instances, we aim to store components nearby so that, upon request arrival, an instance can boot up and respond to that request in the desired response time. Docker [3] [4] could be used to facilitate on-demand provisioning and has seen an increase in popularity. Docker uses operating-system-level virtualization for the deployment of applications inside lightweight software containers. Docker services consist of a layered filesystem (*layers*) where each layer can be reused by other services and stored on different locations. Software containers share the same operating system kernel as the host system, allowing Docker to deploy a running service instance almost instantly.

With multiple services reusing the same layers, the challenge is to find the optimal location for each layer to allow Docker to retrieve layers and deploy an instance quickly so the service can respond to the request within the desired response time. Research on optimal service placement typically minimizes the cost to uphold a certain Service Level Agreement (*SLA*) based on expected demand. However, very little research goes towards the long-tail services with no predictable demand patterns. We speak of long-tail services if the average demand per time unit (Erlang) is less than one for users in a nearby geographical area, which is common for many services used in microservice architectures.

In this paper we propose a service placement algorithm which maximizes the amount of clients we can serve on-demand. Unlike most existing placement algorithms, we focus on long-tail services and consider that clients may connect from any given location to request a service located in the network. Using the layered filesystem introduced by Docker, our algorithm places the layers on nearby storages so that users can connect to a server, download the service layers and deploy a service instance within a desired response time. Docker handles the layer retrieval and instantiation so we only concern ourselves with finding the optimal location for the layers (Fig. 1).

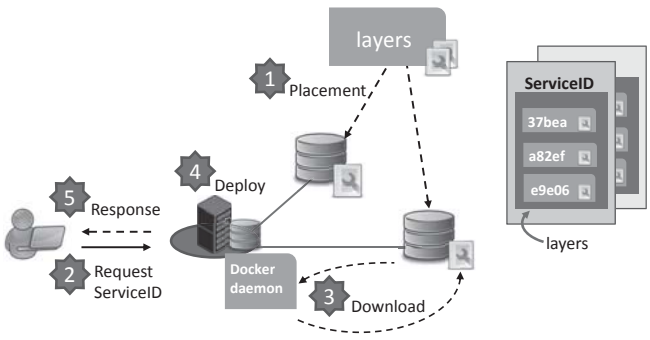

Fig. 1. Our algorithm places Docker image layers on remote storages so that, upon request arrival, Docker retrieves the required service layers, deploys an instance almost instantly and the instance responds to the client, all within the desired response time.

We use integer linear programming (*ILP*) to solve the service placement and selection problem. ILP does not scale well due to heavy memory requirements and is unable to solve very large problems. As we envision a large amount of services and clients in the network, we present algorithms to reduce the scale of the problem while minimizing the performance degradation (*unsatisfied demand*) induced by this information loss.

In the remainder of this paper we present our placement algorithm and research results. In section II we discuss related work to service placement algorithms and also the facility location problem, which is similar to our problem. We then proceed by presenting our problem statement in section III, formulated as an ILP. In section IV we evaluate our simulation results and describe how the scale of our problem can be reduced to solve larger problems. Last, we discuss key aspects of our research that will be tackled in future work in section V.

II. Related work

1. Service placement. As services often reside on cloud sites with dynamic pricing schemes, research on service placement often searches for the placement that minimizes the hosting costs. In [5] the goal is to find the lowest cost while also guaranteeing key performances such as a certain response time. Other approaches directly attempt to maximize the satisfied demand [6]. Typically, these algorithms find a location for a service instance or service component instance assuming that they can be deployed on these locations. Our algorithm aims to find a storage location for each layer so that an instance can be deployed on-demand, avoiding any over-provisioning.

2. Multi-stage selection. Selecting which client connects to which server solves only half our problem, as we also need to determine which storage each layer is downloaded from for each server. This selection problem is similar to recent research on the Facility Location problem. The Facility Location problem is an older concept where the goal is to find the optimal location and amount of facilities to serve as many customers within a certain response time and with minimized costs. However, further research introduced the multi-stage supply chain problem [7] [8] [9], where consumers are not only assigned to a distribution center but distribution centers are also linked to production plants. This problem is also often formulated as an ILP but existing research assumes knowledge of the user demand patterns, which are harder to predict for long-tail services. We solve this problem under the assumption that clients may connect from any given client location and aim to deploy an instance on-demand within the desired time.

III. Pre-deployment layer placement

Our goal is to find the optimal layer placement so that clients can connect to a server, download the service layers and deploy a service instance within a desired response time. In this case we speak of a satisfied client for the requested service. Users may find multiple servers to connect to and there may be multiple locations to download layers from, each resulting in a different response time. To verify that a client can be served on-demand with a certain placement, our algorithm must also select which server a client connects to and which storage locations to download the layers from. If there is a selection which satisfies that user then we must guarantee to find it. When not all clients can be satisfied, our algorithm finds the placement and selection which satisfies the most clients.

We use ILP to formulate and solve the placement and selection problem. ILP considers a set of variables, constraints and an objective function. The values of decision variables have a fixed range for ILP to choose from and define the solution space, while the constraints ensure a feasible solution. The objective function is used to select the best combination of decision variable values which are within the constraints.

A. Problem statement

Variables. Consider a set of clients I, servers J and storages K. The collection of network nodes $N = I \cup J \cup K$ is the union of these three collections. We define the collection of services as S and each service $s \in S$ consists of a set of layers L_S required to deploy an instance of s. The collection of all layers to be placed is denoted L but a layer $l \in L$ may be shared by multiple services and belong to more than one L_S.

Each storage $k \in K$ has a disk space DS_k while the disk space required by layer l is DSR_l. We define the latency between two nodes $i, j \in N$ as $d_{i,j}$ while $H_{i,j}$ is the shortest path length expressed in hops. $D_{j,k,l}$ represents the time to download a layer l from storage k to server j. The weight corresponding to the session traffic between clients and servers is denoted α while β is the weight we assign to the download time of layers between client and storage. We define T_l as the required time to install a layer on the server after downloading it. The desired service response time for service s is D_s.

The decision variables can either have the value one or zero. $P_{i,j,k,l}$=1 if client i connects to server j to download layer l from storage k, otherwise it is 0. If a layer l is placed on storage k then $Y_{k,l}$=1, otherwise 0. We define $Q_{i,j,s}$=0 as a valid selection if there is a server j where client i can download each layer $l \in L_S$ from, as dictated by $P_{i,j,k,l}$. If there is no server j where $P_{i,j,k,l}$=1 for client i and for each layer $l \in L_S$, then $Q_{i,j,s}$=1 represents an invalid selection.

Constraints. The amount of used disk space cannot exceed the available disk space

$$\sum_{l \in L} Y_{k,l} * DSR_l \leq DS_k \ \forall k \in K \tag{1}$$

A client i can only be assigned to download layer l from storage k if layer l is placed on storage k

$$P_{i,j,k,l} \leq Y_{k,l} \ \forall i, j, k, l \tag{2}$$

A client i may only be assigned to at most one server-storage pair to download layer l

$$\sum_{j \in J} \sum_{k \in K} P_{i,j,k,l} \leq 1 \ \forall i, l \tag{3}$$

A valid selection ($Q_{i,j,s} = 0$) must assign all layers $l \in L_S$ to a client i and the client only connects to one server for that service, otherwise $Q_{i,j,s} \geq 1$

$$Q_{i,j,s} \geq \frac{L_S - \sum_{k \in K} \sum_{l \in L_S} P_{i,j,k,l}}{L_S} \ \forall i, j, s \quad (4)$$

Last, the selection must allow a client i to connect to a server j, download layers $l \in L_S$ and deploy them within the desired response time D_s

$$\sum_{j \in J} \sum_{k \in K} \sum_{l \in L_S} P_{i,j,k,l} * (\frac{\alpha * d_{i,j}}{size(L_S)} + \beta * D_{j,k,l} + T_l) \leq D_S \ \forall i, s \quad (5)$$

Note that we divide the latency between client and server by the amount of layers in L_S as the client only connects to the server once for each service request.

Objective function. $Q_{i,j,s} = 0$ only when we made a logical selection where a client is able to download all layers of a service from one server location. When such a selection is found, Eq. 5 guarantees that we can serve the client within the desired response time for that service. Since we want to maximize the amount of clients we can serve, our objective function needs to minimize $Q_{i,j,s}$ over all clients, servers and services:

$$minimize \sum_{i \in I} \sum_{j \in J} \sum_{s \in S} Q_{i,j,s} \quad (6)$$

Note that Eq. 3 guarantees that only one server j will be selected for each client-service pair. If multiple solutions can satisfy the same amount of users, we search for the solution which minimizes the bandwidth. We ensure that the bandwidth factor is smaller than one, which is the smallest unit of $Q_{i,j,s}$, so that the final solution still minimizes Q and thus maximizes the amount of clients we can serve within a desired time. Assuming a stable network, the bandwidth is mostly decided by the path length between source and destination. Therefore, we aim to minimize the path length of all traffic in the final solution.

Consider the following variables:

1) Total hop count in current selection:
   $h_c = \sum_{i,j,k,s,l \in L_S} P_{i,j,k,l} * (\frac{\alpha * H_{i,j}}{size(L_S)} + \beta * H_{j,k})$
2) Total hop count in network:
   $h_t = \sum_{i,j,k,s,l \in L_S} (\frac{\alpha * H_{i,j}}{size(L_S)} + \beta * H_{j,k})$

If multiple solutions can serve the same amount of users within the desired time then we use $\frac{h_c}{h_t} \leq 1$ as tiebreaker so that the solution with the overall least bandwidth usage is chosen. The resulting objective function is:

$$minimize \ \frac{h_c}{h_t} + \sum_{i \in I} \sum_{j \in J} \sum_{s \in S} Q_{i,j,s} \quad (7)$$

IV. Simulation results

In this section we present our simulation results on the performance of our ILP algorithm described in section III-A. ILP is able to find an optimal solution for any problem but requires a lot of system resources when the scale of the problem increases. To increase the scalability of ILP, we present a technique to reduce the scale of the problem while minimizing the unsatisfied demand in section IV-A.

Our algorithm only needs to run once to find the optimal solution for the given network and service conditions. However, as service characteristics tend to be dynamic, we wish to limit the execution time of our algorithm. In section IV-B we discuss the correlation between the time limit and the performance of our algorithm.

We used Brite [10] to generate a Waxmann topology with link latencies between 10-100 milliseconds. We then select the client, server and storage nodes at random and use the generated graph to calculate all the variables required to run our ILP as described in section III-A. The service characteristics were calculated from real-life Docker repositories as explained in section IV-A. All simulations are performed on the iLab.t Virtual Wall [11] using a server with a Hexacore Intel E5645 (2.4GHz) CPU, 24GB RAM, 1x 250GB hard disk and 1-5 gigabit network interface cards.

A. Scalability

We crawled the Docker Hub to determine the characteristics of official Docker images. On average a service has one large base layer and 15 smaller layers, some of those shared with other services. As most of those layers only contain meta data or simple commands, their file size is often less than 1MB, making it easier to group these layers on a storage without exceeding the available disk space. In order to reduce the scale of the problem that ILP has to solve, we studied the effect of combining all these smaller layers into one composite layer. The size, deployment time and download time of this composite layer equals the sum of the respective characteristics of each combined layer. Using this method, each service in our experiment now has one base layer, shared layers and exactly one composite layer. We use the same ILP algorithm to solve the problem but with a reduced dataset. The difference with the original problem is that our placement algorithm is now forced to put all combined layers of a service on the same storage, while the original ILP problem is able to place each layer on a different location. We call this the *Combined Layers* method.

Fig. 2 illustrates the performance of both the original ILP problem and the Combined Layers method where our ILP uses the reduced dataset described above. To facilitate visualizing the performance for a setup with many services, we configure the same desired response time for each service in our network. For each desired response time (X-axis) we measure the performance as the satisfied demand (Y-axis), which equals the amount of clients we can serve on-demand for all services. Compare the Combined Layers and "'ILP - 5 min"' curves in Fig. 2, representing the performance of ILP for both the reduced and original dataset when we limit the execution time to 5 minutes. Combined Layers solves the problem with less degrees of freedom than our original ILP case, as it is forced to place all combined layers of the composite layer on the

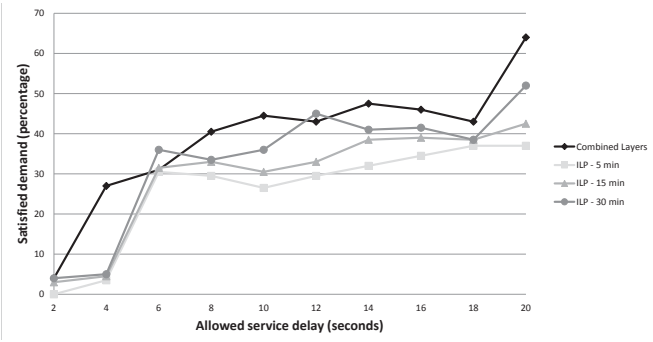

Fig. 2. Performance of our ILP algorithm when using the approximated Combined Layer dataset compared to our original dataset for a problem with 10 clients, 5 servers, 5 storages, 20 services and different time limits.

same storage. However, due to the large decrease of layers per service in the Combined Layers dataset, our algorithm is able to solve the problem more easily and achieve a better solution in the same amount of time as with the original dataset. This shows that although we are using an approximated dataset, the reduced scale allows us to not only minimize the performance loss (*unsatisfied demand*) but in fact achieve an even better performance in a limited time.

Note that each ILP run may be at a different point of the solving process when cut off prematurely, which is why the curves are not monotonically increasing even though the desired response time increases alongside the X-axis.

B. Optimal execution time

As our Combined Layers method uses an approximated dataset of the original problem, it is bound to find a worse or at best an equal solution to our original ILP problem if both algorithms could run without a time limit. However, in section IV-A we illustrated how ILP achieves better results with the Combined Layer dataset than with the original dataset in a limited time. This is useful for large scale problems where we can't wait for the optimal solution and wish to provide a best-effort solution in a short timeframe.

We studied the importance of the time limit by running the original ILP problem for 5, 15 and 30 minutes for the same dataset. Fig. 2 illustrates how we gain roughly 5% more satisfied demand when running our ILP algorithm for 15 minutes compared to 5 minutes. We need to run our ILP algorithm using the original dataset for 30 minutes before we start seeing a better performance than achieved with our Combined Layers dataset in 5 minutes, and only in certain points. This shows that increasing the time limit for the original dataset is not a viable option and that our Combined Layers method is necessary to achieve good results in a shorter timeframe.

V. Conclusion and future work

In this paper we presented a placement and selection algorithm to maximize the amount of clients we can serve on-demand within a desired time. We formulated our problem statement and solved it with ILP. Next, we described how the dataset can be reduced with the Combined Layers approach to facilitate the scalability of our ILP algorithm. Our simulations show that we can achieve better results within a shorter timeframe using an approximated dataset than with the larger original dataset. Last, we evaluated the influence of the time limit on the performance of our algorithm and observed that a large time limit is required to see significant improvement. We conclude that the Combined Layers method is the best performing algorithm for larger scale problems where a best-effort solution must be found in a short timeframe.

Future research will focus on more methods to reduce the scale of our dataset while minizing the performance loss due to approximations. Also, we will not only guarantee a first response time but also a desired latency between client-server to optimize the session traffic after the first response. Additionally, We will verify our simulation results on real-life topologies available in the Topology Zoo [12] dataset, an ongoing project to collect network topology data worldwide. Using real-life topologies we will be able to show the importance of our research in real-life scenarios.

Acknowledgments

This project was partly funded by the UGent BOF-GOA project "'Autonomic Networked Multimedia Systems"', by the FWO-V project "'SPEC: Intelligent SuPer-Elastic Clouds"' and by the 7th Framework Programme of the European Commission through the FUSION project under grant agreement no. 318205.

References

[1] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[2] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[3] J. Fink, "Docker: a software as a service, operating system-level virtualization framework," *Code4Lib Journal*, vol. 25, 2014.

[4] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[5] Q. Zhang *et al.*, "Dynamic service placement in geographically distributed clouds," *Selected Areas in Communications, IEEE Journal on*, vol. 31, no. 12, pp. 762–772, December 2013.

[6] J. Famaey *et al.*, "Network-aware service placement and selection algorithms on large-scale overlay networks," *Computer Communications*, vol. 34, no. 15, pp. 1777–1787, 2011.

[7] T. Wu *et al.*, "A lagrangean relaxation approach for a two-stage capacitated facility location problem with choice of depot size," in *Networking, Sensing and Control (ICNSC), 2015 IEEE 12th International Conference on*, April 2015, pp. 39–44.

[8] J. Li *et al.*, "Lower and upper bounds for a two-stage capacitated facility location problem with handling costs," *European Journal of Operational Research*, vol. 236, no. 3, pp. 957–967, 2014.

[9] A. Syarif *et al.*, "Study on multi-stage logistic chain network: a spanning tree-based genetic algorithm approach," *Computers & Industrial Engineering*, vol. 43, no. 1, pp. 299–314, 2002.

[10] (2013) Brite: Boston university representative internet topology generator. [Online]. Available: http://www.cs.bu.edu/brite/

[11] (2013) ilab.t virtual wall | internet based communication networks and services. [Online]. Available: http://www.ibcn.intec.ugent.be/content/ilabt-virtual-wall

[12] S. Knight *et al.*, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765–1775, 2011.